Enhancing Efficiency and Security in Unbalanced PSI-CA Protocols through Cloud Computing and Homomorphic Encryption in Mobile Networks

Wuzheng Tan , Shenglong Du * , Jian Weng

Posted Date: 7 May 2024

doi: 10.20944/preprints202405.0307.v1

Keywords: private set intersection cardinality; cryptographic; commutative encryption; Cuckoo filter; cloud computing

Article

Enhancing Efficiency and Security in Unbalanced PSI-CA Protocols through Cloud Computing and Homomorphic Encryption in Mobile Networks

Wuzheng Tan [1,2], Shenglong Du [1,2,*] and Jian Weng [1,2]

[1] College of Cyber Security, Jinan University, Guangzhou 510632
[2] Guangdong Key Laboratory of Data Security and Privacy Preserving
[*] Correspondence: jndsl@stu2021.jnu.edu.cn

Abstract: Private Set Intersection Cardinality(PSI-CA) is a cryptographic method in secure multi-party computation that allows entities to identify cardinality of the intersection without revealing their private data.Traditional approaches assume similar-sized datasets and equal computational power, overlooking practical imbalances.In real-world applications, dataset sizes and computational capacities often vary, particularly in the Internet of Things and mobile scenarios where device limitations restrict computational types. Traditional PSI-CA protocols are inefficient here, as computational and communication complexities correlate with the size of larger datasets. Thus, adapting PSI-CA protocols to these imbalances is crucial.This paper explores unbalanced scenarios where one party (the receiver) has a relatively small dataset and limited computational power, while the other party (the sender) has a large amount of data and strong computational capabilities.This paper, based on the concept of commutative encryption, introduces Cuckoo filter, cloud computing technology, homomorphic encryption, among other technologies, to construct three novel solutions for unbalanced Private Set Intersection Cardinality (PSI-CA): an unbalanced PSI-CA protocol based on Cuckoo filter, an unbalanced PSI-CA protocol based on single cloud assistance, and an unbalanced PSI-CA protocol based on dual cloud assistance. Depending on performance and security requirements, different protocols can be employed for various applications.

Keywords: private set intersection cardinality; cryptographic; commutative encryption; Cuckoo filter; cloud computing

1. Introduction

1.1. Background

In today's digital age, data privacy and security have become critically important issues worldwide. With technological advancements and explosive growth in data volumes, individuals and institutions face unprecedented challenges in protecting their privacy. Privacy computing technologies have emerged in response to these challenges, enabling the secure computation and analysis of data without exposing the details of personal information. This is crucial for driving data-driven innovation and services while safeguarding personal privacy and data protection.

Private Set Intersection (PSI) technology is a key technique in the field of privacy computing. It allows two or more parties to identify the common elements in their data sets without revealing any other non-shared data. This technology is highly useful in multiple application scenarios, such as cross-institutional data cooperation, fraud detection, and private contact discovery, without compromising user privacy. It has been applied in various fields, including genetic testing of fully sequenced human genomes [1], private contact discovery [2], and botnet detection [3]. This study investigates the cardinality of private data set intersections between two parties, which is an essential aspect of two-party computation (2PC) tasks. Specifically, it involves a sender and a receiver who aim to collaboratively determine the number of common elements in their private data sets. Throughout this process, only the receiver obtains the cardinality of the set intersection, while the sender remains unaware of it. The topic of private set intersection cardinality is widely researched due to its significant practical applications

 doi:10.20944/preprints202405.0307.v1

1.2. Motivation

While the traditional Private Set Intersection Cardinality (PSI-CA) protocols has been extensively explored in research, real-world applications continue to present unique challenges.Traditional approaches assume similar-sized datasets and equal computational power, overlooking practical imbalances.In real-world applications, dataset sizes and computational capacities often vary, particularly in the Internet of Things and mobile scenarios where device limitations restrict computational types. Traditional PSI-CA protocols are inefficient here, as computational and communication complexities correlate with the size of larger datasets. Thus, adapting PSI-CA protocols to these imbalances is crucial.

A compelling example is found in the collaboration between major medical institutions and small health app developers. In this scenario, a large medical institution with extensive patient data and robust computational capabilities collaborates with a small app developer who possesses minimal user data and limited computational resources. The primary goal is to analyze the coverage of health app users within the extensive patient database to assess market penetration and potential partnership opportunities. For instance, the medical institution might want to determine how many of its patients are using the health app to consider recommending it more broadly or collaborating on new features.

To address the challenges outlined above, this paper delves into the unbalanced Private Set Intersection-Cardinality (PSI-CA) protocols and proposes three innovative unbalanced PSI-CA protocols. In practical applications, different solutions can be chosen based on varying performance and security requirements.

1.3. Main Work

1. To address the performance shortcomings of traditional PSI-CA protocols in the face of significant differences in dataset sizes between participants, this paper introduces the first protocol, which is the unbalanced PSI-CA protocol based on Cuckoo filter. This protocol successfully constructs the first unbalanced private intersection cardinality protocol of this article by integrating exchange encryption technologies with Cuckoo filter functionalities for private information retrieval, followed by experimental analysis.
2. To alleviate the computational and storage burden on small health app developer in the first protocol, the paper further proposes an unbalanced PSI-CA protocol based on single cloud assistance and conducts experimental analysis. This strategy effectively migrates computational and storage tasks to cloud services, significantly optimizing resource utilization efficiency.
3. To safeguard against data leakage risks inherent in the unbalanced PSI-CA protocol based on single cloud assistance which cannot resist collusion attacks, the paper further designs an unbalanced PSI-CA protocol based on dual cloud assistance. By employing homomorphic encryption and other security technologies, this scheme resolves potential data leakage risks in the single-cloud protocol while effectively preventing potential collusion attacks.
4. Based on the unbalanced PSI-CA protocol based on dual cloud assistance, this paper also designs the PSI-CA network and establishes corresponding data update strategies, significantly enhancing the practicality of the protocol.

2. Related Works

2.1. Design Framework of Private Set Intersection Protocol

2.1.1. Design Framework Based on Public Key Encryption

The basic idea behind early private set intersection protocols is to encrypt data elements and then perform comparison operations on the encrypted data. The most widely used technique in this method is homomorphic encryption: the sender encrypts their dataset and sends it to the receiver. The receiver processes these ciphertexts using the properties of homomorphic encryption and returns the results to the sender. The sender then decrypts these results using their own private key to obtain

 doi:10.20944/preprints202405.0307.v1

the intersection of the datasets. This public-key-based method generally relies on three main security assumptions [4]:

1. Based on Diffie-Hellman (DH) theory: Meadows [5] used the DH key exchange mechanism, which is based on the discrete logarithm problem, to implement a PSI protocol. In contrast, Huberman [6] and his team explored the use of elliptic curve cryptography in PSI, noting its significant advantages in security and efficiency compared to traditional discrete logarithm-based PSI methods.
2. Based on the RSA assumption: DeCristofaro and others [7] developed a semi-honest PSI protocol using RSA blind signature technology based on the integer factorization problem. Another study [8] showed that PSI schemes based on discrete logarithm cryptography demonstrated higher efficiency compared to those based on integer factorization cryptography.
3. Based on homomorphic encryption: Freedman and his team [9] innovatively represented elements as roots of polynomials and encrypted the coefficients of these polynomials using Paillier homomorphic encryption technology, combined with zero-knowledge proofs, to implement a two-party PSI protocol resistant to malicious attacks. In 2016, Freedman et al. [10] further improved computational efficiency through the ElGamal encryption mechanism and reduced the protocol's computational complexity using Cuckoo Hash technology [4]. Abadi et al. [11] introduced a set representation method based on point-value pairs of d-degree polynomials, implemented through the Paillier encryption scheme, reducing the multiplication complexity from $O(d^2)$ to $O(d)$ [4]. Kissner and other researchers [12] adopted different polynomial representation methods, significantly reducing computational costs to be linearly proportional to the number of participants. Jarecki and others [13] used additive homomorphic encryption and zero-knowledge proofs to implement pseudorandom functions (PRF). Hazay and others [14] developed an additive homomorphic encryption scheme that supports threshold decryption for implementing multi-party semi-honest PSI protocols. Dou Jiawei and others [15] combined Paillier encryption to propose a PSI protocol based on the formula for calculating the area of triangles and rational number encoding.

Public key encryption-based Private Set Intersection (PSI) schemes typically feature fewer communication rounds and are suitable for environments with strong computational capabilities. However, in practice, communication bandwidth and time complexity often pose significant constraints [4].

2.1.2. Design Framework Based on Garbled Circuits

Garbled circuit technology can transform any function into a Boolean circuit, thereby securely computing the function. Early methods based on universal circuits, like the DPSZ scheme [16], demonstrated how to use arithmetic circuits to solve the set intersection problem: the circuit builder encrypts the circuit gates using a symmetric key, then creates a garbled circuit and sends it to the circuit evaluator. The evaluator decrypts specific paths in the garbled circuit to obtain the intersection results, while being unable to access other paths in the circuit. As the circuit depth increases, its construction complexity also increases. Additionally, PSI protocols based on this circuit design can also perform various symmetric function operations, such as calculating the threshold intersection, the number of intersection elements, and their sum. For adversaries under semi-honest conditions, there are two types of garbled circuits: the Yao [17] protocol and the GMW [18] protocol. Pinkas et al. [19–21] and Chandran et al.based on hash storage structures and GMW circuits, implemented a more efficient OPRF circuit PSI scheme through private membership tests, reducing the number of comparisons and the depth of circuit equivalence comparisons. Meanwhile, Huang [22] and others created a semi-honest secure disordered circuit PSI scheme through the combination of Yao circuits, performing equivalence tests and specific sorting on adjacent elements. Despite these advantages, these methods still require additional key calculations and communication processes, such as key exchanges between participants.

 doi:10.20944/preprints202405.0307.v1

2.1.3. Design Framework Based on Oblivious Transfer

Oblivious Transfer (OT) [23] is a cryptographic protocol that allows a sender to transmit information to a receiver without revealing any private information. In the OT protocol, the sender has two options, but the receiver can only obtain information about one of them without access to the other, and the sender does not know which option the receiver has chosen. OT is widely used in many secure areas due to its cryptographic robustness and privacy features. Its applications include secure protocol negotiation, secure online auction systems, and secure voting systems. In 2013, Dong [24] et al. proposed a new data structure—the garbled Bloom filter (GBF)—and based on the GBF and OT extension, they introduced a PSI protocol. This protocol utilized efficient symmetric encryption operations and could handle billions of elements. However, this protocol faced two issues: one is that the malicious sender might send incorrect shared information, and the other is that the input datasets are not independent. To address these problems, in 2016, Rindal and Rosulek [25] proposed a new randomized garbled Bloom filter using the "cut-and-choose" technique. They successfully implemented a two-party malicious model PSI protocol. Subsequently, Zhang et al. [26], based on this scheme, further proposed and implemented a multi-party PSI protocol, ensuring malicious security in the presence of two non-colluding servers, with computational and communication costs depending on the number of participants. Pinkas et al. [27] based on the OOS17-OT [28] protocol, built a maliciously secure PSI protocol. Rindal [29] et al. based on the semi-honest secure Schoppmann et al. [30] protocol and the maliciously secure Weng et al. [31] protocol, respectively proposed maliciously secure and semi-honest secure PSI protocols. Overall, PSI protocols based on oblivious transfer typically feature lower computational and communication overhead.

2.2. PSI-CA

In the research of set intersection cardinality, [32] put forward a two-party protocol based on the Bloom filter and ElGamal encryption scheme, the trick here is to count how many common zero-bit in their Bloom filters. Thus the cardinality of set intersection can be quickly figured out by a formula provided in [33]. The proposal in [34] was analogous to [32], which employed the Goldwasser–Micali encryption algorithm [25] to encrypt the Bloom filter entries.Cristofaro et al. [35]designed the first PSI-CA protocol with linear complexity $O(n_y + n_x)$ based on the Decisional Diffie-Hellman (DDH) assumption in the random oracle model (ROM) [36]. However, for the computation cost, both the sender and the receiver need to compute two exponentiations for each item.

3. Related Theories and Technologies

3.1. Multi-Party Secure Computation Security Model

The mathematical concept of Multi-Party Computation (MPC) involves several participants (such as $P_1, P_2, \ldots, P_n$) each holding private input data (x_i). These participants collaboratively execute a computation of the function $f(x_1, x_2, \ldots, x_n)$ with the goal of ensuring that each participant can only access their own computational results, while being unable to ascertain the inputs and results of others. There are generally two security models employed in secure multi-party computation protocols [37,38]:

1. **Semi-honest model:** In this model, participants adhere to the protocol's execution rules but may attempt to gather other participants' inputs, outputs, and any accessible information during the execution of the protocol. This model assumes that the participants do not deviate from the established procedural rules but will use all available information to deduce the private data of others.
2. **Malicious adversary model:** Unlike the semi-honest model, the malicious adversary model accounts for the possibility that attackers may manipulate a subset of the participants to perform illicit actions, such as submitting incorrect input data or maliciously altering data to steal the private information of honest participants. Malicious adversaries might also disrupt the protocol

 doi:10.20944/preprints202405.0307.v1

by intentionally terminating its execution or by refusing to participate, thus preventing the protocol's completion.

The security model considered in this paper is the semi-honest security model.

3.2. Cuckoo Filter

Determining whether a particular element belongs to a given set is a common problem in computer science, with widespread applications in bioinformatics, machine learning, computer networks, the Internet of Things, and database systems [39]. Filter data structures such as Bloom filters and Cuckoo filters can approximately determine if an element is part of a specified set and have been extensively applied in network routing [40], information retrieval, file merging [41], spam detection [42], and distributed systems [43].

Filter data structures are used to approximately ascertain if an element belongs to a specific set. In essence, for a given set S and a query element x, the filter can approximately inform the query whether "x is in S". "Approximately" here implies that if x is actually not in S, the filter has a small error probability p of wrongly indicating that "x is in S"; however, if x is indeed in S, the filter will always correctly return that "x is in S". Filter data structures sacrifice some query accuracy to enhance space and time efficiency. Unlike data structures that require storing complete information of each element for precise queries, filters approximate the presence of an element solely through partial information such as hash values or "fingerprints". Based on this principle, existing filter data structures are mainly categorized into two types: one type uses bit arrays as in Bloom filters; the other type, exemplified by Cuckoo filters, is based on element "fingerprints".

The Cuckoo filter [44] is an advanced retrieval structure made up of multiple buckets, each capable of containing several bits. Compared to Bloom filters, Cuckoo filters offer the significant advantage of supporting deletion of elements and having higher space efficiency. With equal storage space, Cuckoo filters can achieve more accurate search results and shorter search times. When querying an element, the time complexity for Cuckoo filters is $O(1)$, meaning constant time complexity. This indicates that the execution time for query operations does not increase with the number of elements in the filter, an important performance feature of the Cuckoo filter design. In this paper, Cuckoo filter are used to store data on large medical institution.

3.3. Paillier Homomorphic Encryption

Homomorphic encryption is an encryption technology that allows computations to be performed on encrypted data and to obtain encrypted results, which, when decrypted, are consistent with the results obtained by performing the same computations directly on the original data. This means that homomorphic encryption enables data to be processed and analyzed without revealing any content. It is an important technology for protecting online privacy, allowing cloud computing services to perform complex data processing tasks on users' encrypted data without accessing the actual data.

Paillier homomorphic encryption is a public-key cryptosystem that specifically supports homomorphic addition operations on encrypted data. The applications of the Paillier encryption scheme are extensive, and it can be used to protect the privacy and security of data. For example, in distributed computing, the Paillier encryption scheme can be used to encrypt data and transmit it to various nodes for processing, ensuring the security and privacy of the data. Furthermore, the Paillier encryption scheme can also be used to implement homomorphic secret sharing, private set intersection, and other application scenarios. Overall, the Paillier encryption scheme is an efficient homomorphic encryption scheme with a wide range of application prospects. This paper uses the Paillier cryptosystem in its final scheme. The homomorphic properties utilized in this paper are as follows. The final scheme is based on these two features:

1. **Additive Homomorphism**: If $c_1 = \text{Enc}(m_1)$ and $c_2 = \text{Enc}(m_2)$, then $\text{Dec}(c_1 \cdot c_2 \mod n^2) = m_1 + m_2$. This allows for performing addition operations on ciphertexts without needing to decrypt them first.

 doi:10.20944/preprints202405.0307.v1

2. **Scalar Multiplication Homomorphism**: If $c = \text{Enc}(m)$, then $\text{Dec}(c^k \mod n^2) = k \cdot m$. This means that it is possible to perform multiplication operations between a ciphertext and a plaintext scalar without decryption.

This paper will utilize homomorphic encryption technology to construct the third protocol of this paper: Unbalanced PSI-CA Protocol Based on Dual Cloud Assistance.

4. PSI-CA Protocol Constructed Based on DH Key Exchange Mechanism

Before proposing the first unbalanced PSI-CA protocol of this paper, we introduce Cristofaro' PSI-CA protocol constructed based on the DH key exchange mechanism [35].The specific process is as follows:

4.1. Protocol Process

4.1.1. Exchange and Computation Stage

1. **Receiver Data Encryption**: The receiver encrypts $H(y_i)$ with its private key β, obtaining $H(y_i)^\beta$ and sends it to the sender.
2. **Sender Computation**: Upon receiving $H(y_i)^\beta$, the sender applies its private key α to compute $(H(y_i)^\beta)^\alpha$ and shuffles it before sending it back to the receiver.
3. **Sender Data Encryption**: The sender encrypts $H(x_i)$ with its private key α, resulting in $H(x_i)^\alpha$ and sends it to the receiver to facilitate the computation of the intersection cardinality.

4.1.2. Cardinality Calculation Stage

1. **Receiver Decryption and Computation**: The receiver uses the inverse of β to decrypt $(H(y_i)^\beta)^\alpha$ to retrieve $H(y_i)^\alpha$. By comparing $H(x_i)^\alpha$ with $H(y_i)^\alpha$, the receiver can calculate the cardinality of the intersection between the two sets.

4.2. Experimental Analysis

For this protocol, experiments were conducted and the runtime was recorded for various combinations of dataset sizes, as shown in Table 1.

Table 1. Runtime of the PSI-CA Protocol Constructed Based on DH Key Exchange Mechanism

Cardinality of Dataset from Participant One	Cardinality of Dataset from Participant Two	Protocol Runtime (seconds)
2^{10}	2^{15}	1.8095
2^{10}	2^{17}	7.3003
2^{10}	2^{20}	56.1277
2^{10}	2^{25}	1859.9520
2^{15}	2^{15}	5.2207
2^{15}	2^{17}	10.0672
2^{15}	2^{20}	63.4835
2^{15}	2^{25}	1886.3966
2^{17}	2^{17}	20.7044
2^{17}	2^{20}	71.9252
2^{17}	2^{25}	1977.5657
2^{20}	2^{20}	170.3074
2^{20}	2^{25}	2054.2694

Through the experimental data, an important phenomenon can be observed. Table 1 shows the estimated runtime of the above protocol under different data volume levels. For example: Initially,

 doi:10.20944/preprints202405.0307.v1

when the number of elements in Participant One's dataset is 2^{10} and Participant Two's dataset is 2^{15}, the runtime of the protocol is 1.745 seconds. In this case, there is a noticeable imbalance between the smaller side (Participant One) and the larger side (Participant Two). Now, if we expand the number of elements in Participant One's dataset (originally the side with fewer elements) to 2^{15}, while keeping Participant Two's dataset size constant at 2^{15}, the runtime increases to 4.925 seconds, approximately three times the original. This indicates that although the runtime increases when the datasets are balanced, the increase is limited. However, if we keep Participant One's dataset size at 2^{10} and increase Participant Two's dataset size to 2^{20} (the same scale of change), the runtime dramatically increases to 55.267 seconds, about 31 times the initial condition. This phenomenon shows that in unbalanced dataset conditions, increasing the number of elements in the larger dataset significantly affects the efficiency of the protocol.

These results reveal the importance of dataset balance in maintaining efficiency during the implementation of this protocol. Unbalanced datasets not only lead to extended runtimes but can also cause low resource utilization and delays in processing. However, in practical applications, when two parties want to get private set intersection's cardinality, their sets are often unequal and with a significant gap. Therefore, the current situation requires the design of a new protocol to eliminate the impact of dataset size imbalance on protocol efficiency.

4.3. Summary of This Chapter

This chapter explores the PSI-CA Protocol built on the Diffie-Hellman (DH) key exchange mechanism, detailing its processes and experimental analysis. Initially, the protocol employs a DH mechanism for securing data exchanges between two parties, outlined in specific stages: data encryption by the receiver, computation and further encryption by the sender, followed by decryption and intersection cardinality computation by the receiver.

The experimental analysis section provides a practical examination of the protocol's runtime across varying dataset sizes, demonstrating that imbalances significantly affect efficiency. As the dataset sizes diverge, particularly when a smaller dataset is compared with a rapidly increasing larger dataset, the protocol's runtime escalates dramatically.

Therefore, there is an urgent need to design a new protocol to alleviate the adverse effects of dataset size imbalance on the performance of the PSI-CA protocol.

5. Unbalanced PSI-CA Protocol Based on Cuckoo Filter

Although Cristofaro' PSI-CA protocol [35] constructed based on the DH key exchange mechanism provides an effective way to compute the intersection's cardinality of two datasets, especially under the premise of protecting participants' data privacy, this paper observes that its efficiency is significantly impacted when dataset sizes are extremely unbalanced. In particular, as shown in Table 1, the runtime increases significantly as the size of the larger dataset increases, reflecting the performance limitations of Cristofaro' PSI protocol when dealing with unbalanced datasets.

In order to overcome these limitations, the first protocol proposed in this paper adopts a different technical strategy, which effectively reduces the computational burden under unbalanced conditions by introducing Cuckoo filter. This not only optimizes the data processing process, but also improves the overall operational efficiency. In the new protocol, the increase in run time is not as dramatic as that in the Cristofaro' PSI-CA protocol [35] based on the DH key exchange, even with unbalanced data set sizes, this allows for more efficient and balanced data processing. This improvement is particularly important for data sets of different sizes frequently encountered in practical applications.

Therefore, based on the above introduction, as shown in Figure 1, this paper first proposes a PSI-CA protocol based on the discrete logarithm problem difficulty and the correctness (high false positive rate) of Cuckoo filter. This protocol is divided into two phases, the specific details are as follows

 doi:10.20944/preprints202405.0307.v1

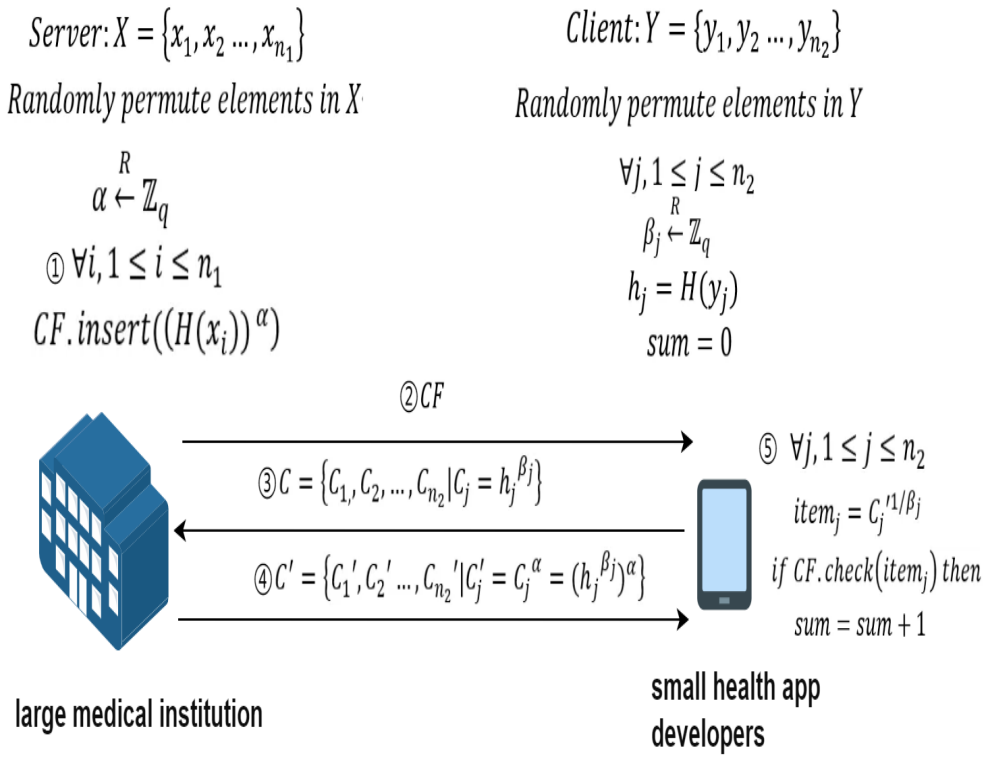

Figure 1. Unbalanced PSI-CA Protocol Based On Cuckoo Filter

5.1. Definition of Main Participants and Related Symbols

1. large medical institution represents the party with a larger dataset and greater computational and storage capabilities.
2. small health app developer represents the party with a smaller dataset and lesser computational and storage capabilities.
3. X and Y represent the dataset of the large medical institution and the small health app developer respectively.
4. α represents the private key of the large medical institution in the Diffie-Hellman encryption algorithm.
5. β_j represents the random number generated by the small health app developer for the Diffie-Hellman encryption algorithm.
6. H represents the hash function negotiated by the small health app developer and large medical institution for use.
7. CF represents Cuckoo Filter,$CF.insert$ represents the operation of adding an element to the Cuckoo filter,$CF.check$ represents the operation of checking whether a specific element exists in the filter.
8. X_i represents the i-th element of set X. Similarly, Y_i, C_i, etc., also represent similar meanings.
9. $C = \{c_1, c_2, \ldots, c_n\}$ represents the set containing n_2 ciphertexts sent by the small health app developer to the large medical institution.
10. C' represents the set containing n_2 ciphertexts sent by the large medical institution to the small health app developer.
11. $item_j$ represents the result obtained through a series of exchange and decryption operations, used to retrieve the filter.
12. sum represents the cardinality of the intersection between the two parties.

5.2. Protocol Process

The protocol is divided into two phases: the preprocessing phase and the intersection phase, with specific details as follows.

 doi:10.20944/preprints202405.0307.v1

5.2.1. Preprocessing

In the preprocessing phase, the small health app developer and the large medical institution need to perform a series of preparatory work to ensure the security and efficiency of subsequent interactions. The specific steps are as follows:

1. Security parameter negotiation: The small health app developer and the large medical institution agree on the large prime number q used in the DH encryption algorithm and the hash function H used.
2. Large Medical Institution Generates Private Key: The large medical institution generates its own private key α, used for the Diffie-Hellman (DH) encryption algorithm.
3. Data Scrambling: The small health app developer and the large medical institution scramble their own datasets Y and X for randomization, enhancing data privacy and security.
4. Small Health App Developer Data Preprocessing: The small health app developer calculates $h_j = H(y_j)$ and generates n_2 random numbers β_j, used for the Diffie-Hellman (DH) encryption algorithm.
5. Creation of Cuckoo Filter: The large medical institution generates a Cuckoo filter CF by using the operation $CF.insert((H(x_i))^\alpha)$, and sends the filter CF to the small health app developer for private set intersection queries with privacy protection.

5.2.2. Cardinality Calculation

In the cardinality calculation phase, the small health app developer and the large medical institution perform a series of carefully designed encryption and decryption operations to blind the small health app developer's elements securely and compute the intersection's cardinality of the two sets. The specific operations are as follows:

1. Element Blinding and Interactive Encryption Operations: The small health app developer and the large medical institution interact through a series of asymmetric encryption and decryption operations to blind the small health app developer's elements. Specifically, the small health app developer calculates $C_j = h_j^{\beta_j}$ and sends C to the large medical institution. The large medical institution uses its private key α to compute $C'_j = C_j^\alpha$ and sends C' back to the small health app developer.
2. Cardinality Computation: After receiving C', the small health app developer checks whether they belong to the filter CF through the check operation $CF.check$, thereby calculating the cardinality of the intersection of the sets. Specifically, after receiving $H(X)^{r\alpha}$ sent by the large medical institution, the small health app developer computes $item_j = C_j'^{1/\beta_j}$ and uses the result to query the filter CF to obtain the intersection's cardinality sum.

5.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, then $item_j = C_j'^{1/\beta_j} = C_j^{\alpha * 1/\beta_j} = \left(h_j^{\beta_j}\right)^{1/\beta_j * \alpha} = h_j^\alpha = H(x_i)^\alpha = H(y_j)^\alpha$.

Thus, through this scheme, the small health app developer can accurately obtain the cardinality of the intersection of both parties.

5.4. Security Analysis

This section will analyze the security of the protocol in detail, mainly its ability to protect the privacy of both parties.

Firstly, considering that the protocol utilizes the Diffie-Hellman (DH) key exchange mechanism to blind the small health app developer's elements, this process's security is based on the difficulty of solving the One-More-Gap-Diffie-Hellman (OMGDH) problem. Since the DH mechanism ensures that even in public communication channels, unauthorized parties cannot decipher the exchanged secret information, the small health app developer's data is protected during transmission to the server. The

 doi:10.20944/preprints202405.0307.v1

server uses a private key to process the received data and returns the results to the small health app developer, this process likewise ensures the security and privacy of the data server.

Secondly, the protocol's use of Cuckoo filter, while efficiently supporting insertion and query operations, its false positive characteristics mean that even if some non-intersecting elements are mistakenly identified as belonging to the intersection, it does not reveal the exact set membership information. This feature provides additional privacy protection to some extent, as even in the event of a false positive error, attackers cannot determine whether a specific element truly exists in the other party's set.

Furthermore, through the interactive computations between the small health app developer and the large medical institution, the protocol ensures that only elements common to both parties can be accurately identified. The small health app developer checks the data returned by the server against its own dataset to ultimately determine the intersection's cardinality.

In summary, based on the blinding process using the Diffie-Hellman mechanism and the use of Cuckoo filter, this protocol can accurately calculate the cardinality of the intersection between two sets while protecting the participants' privacy. It is worth noting that neither party can obtain the specific intersection elements.

5.5. Experimental Analysis

For this protocol, experiments were conducted, and the runtime was recorded for various combinations of data volumes, as shown in Table 2. The table also compares the runtime of Cristofaro' PSI-CA protocol constructed based on the DH key exchange mechanism [35]. Since preprocessing can be completed offline, the runtime of the unbalanced PSI-CA protocol based on Cuckoo filter refers to the total time of the the cardinality calculation process. The original protocol refers to the PSI-CA protocol constructed based on the DH key exchange mechanism, and the new protocol refers to the unbalanced PSI protocol based on Cuckoo filter.

Table 2. Comparison of running time of PSI-CA protocol based on DH key exchange mechanism and Cuckoo Filter based

Cardinality of Dataset from Participant One	Cardinality of Dataset from Participant Two	Original Protocol Runtime (seconds)	New Protocol Runtime (seconds)
2^{10}	2^{15}	1.8095	0.1685
2^{10}	2^{17}	7.3003	0.1663
2^{10}	2^{20}	56.1277	0.1641
2^{10}	2^{25}	1859.9520	0.1840
2^{15}	2^{15}	5.2207	5.2725
2^{15}	2^{17}	10.0672	5.5104
2^{15}	2^{20}	63.4835	5.2807
2^{15}	2^{25}	1886.3966	5.5166
2^{17}	2^{17}	20.7044	21.1474
2^{17}	2^{20}	71.9252	21.2865
2^{17}	2^{25}	1977.5657	21.9713
2^{20}	2^{20}	170.3074	171.5354
2^{20}	2^{25}	2054.2694	186.9644

Through the experimental data, this paper can observe several key phenomena. First, when the cardinality of the smaller dataset (number of elements) remains constant while the number of elements in the larger dataset increases rapidly, it is observed that the runtime of the protocol does not change much, remaining consistent. This indicates that although the size of the large dataset increases dramatically, the efficiency of the protocol is not significantly affected, thereby proving that the design of this protocol can effectively mitigate the negative impact of dataset size imbalance on protocol efficiency. Especially, the overall runtime of the protocol is more related to the cardinality of the smaller dataset and has very low relevance to the cardinality of the larger dataset.

 doi:10.20944/preprints202405.0307.v1

At the same time, this experiment also found that under balanced dataset conditions, the runtime of the PSI-CA protocol based on the DH key exchange mechanism and the unbalanced PSI-CA protocol based on Cuckoo filter does not differ significantly. This indicates that when the sizes of the sets are similar, both protocols can exhibit comparable performance, providing an efficient solution.

5.6. Summary of This Chapter

This chapter presents the development and analysis of an Unbalanced PSI-CA Protocol that utilizes Cuckoo filter to efficiently handle datasets with significant size disparities. The protocol is designed to overcome the limitations observed in traditional PSI-CA protocols such as Cristofaro's, which struggles with efficiency under unbalanced conditions.

Experimental analyses demonstrate the protocol's robustness, showing minimal runtime increases even as dataset sizes grow significantly, which marks a substantial improvement over traditional methods. The protocol proves particularly effective in real-world scenarios where dataset imbalances are common, providing a reliable solution that ensures privacy and efficiency.

In essence, this chapter confirms the efficacy of integrating Cuckoo filter into PSI-CA protocols, offering enhanced performance and security, making it a valuable addition to the field of data privacy and secure computation.

However, in this protocol, the receiver still has to bear the burden of complex cryptographic computations and storing the Cuckoo filter. The next chapter will focus on optimizing this aspect.

6. Unbalanced PSI-CA Protocol Based on Single Cloud Assistance

The previous chapter has proven that the unbalanced PSI-CA protocol based on Cuckoo filter is more suitable for practical scenarios, especially under unbalanced conditions, this protocol effectively resolves the performance limitations of the PSI protocol constructed using the DH key exchange mechanism in handling unbalanced datasets. However, there is still room for improvement in this protocol. It is observed that in this protocol, small health app developer (receiver) need to store filter and perform complex cryptographic operations, which can be a significant burden for mobile devices with limited computing power and storage space. A series of encryption operations and storing filter received from the other party becomes a heavy load. To address this issue, it is considered to transfer most of the receiver's computational and storage tasks to cloud servers. By delegating tasks to cloud servers, receiver can significantly reduce computational and storage pressure, especially for small health app developer with limited capabilities.

This section will introduce cloud computing technology, which allows small health app developer with limited computing power and storage space to outsource their private data and request cloud platforms to perform related computations. Currently, whether for individual users or large enterprises, entrusting data storage and computation tasks to cloud services has become a common practice. Based on the introduction above, as shown in Figure 2, this chapter proposes a second unbalanced PSI-CA protocol.

 doi:10.20944/preprints202405.0307.v1

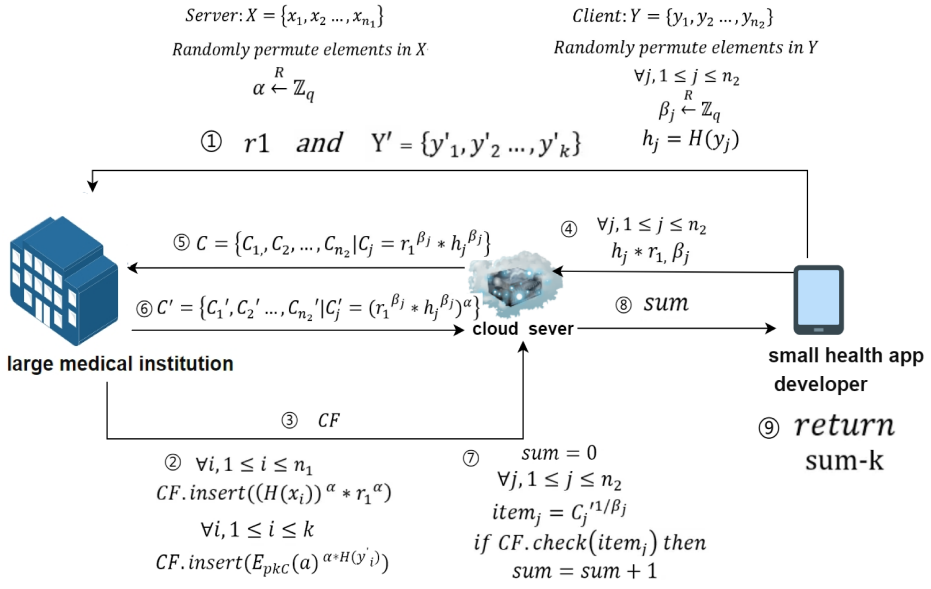

Figure 2. Unbalanced PSI-CA Protocol Based On Single Cloud Assistance

6.1. Definition of Main Participants and Related Symbols

1. large medical institution represents the party with a larger dataset and greater computational and storage capabilities.
2. small health app developer represents the party with a smaller dataset and lesser computational and storage capabilities.
3. cloud server: Represents an auxiliary server that assists the receiver in obtaining intersection's cardinality operations, undertaking most of the computational and storage pressures.
4. X and Y represent the dataset of the large medical institution and the small health app developer respectively.
5. Y' represents the obfuscated dataset sent by the small health app developer to the large medical institution, used to confuse the cloud server and prevent it from obtaining the accurate cardinality of the intersection.k represents the cardinality of the set Y'.
6. α represents the private key of the dlarge medical institution in the Diffie-Hellman encryption algorithm.
7. r_1 represents the random number generated by the small health app developer, used to blind the data.
8. β_j represents the random number generated by the small health app developer for the Diffie-Hellman encryption algorithm.
9. H represents the hash function negotiated for use by the small health app developer and large medical institution.
10. CF represents the Cuckoo Filter, $CF.insert$ represents the operation to add an element to the Cuckoo filter, $CF.check$ represents the operation to check if a specified element exists in the filter.
11. X_i represents the i-th element of the set X. Similarly, Y_i, C_i, etc., also represent similar meanings.
12. $C = \{c_1, c_2, \ldots, c_n\}$ represents the set of n_2 ciphertexts sent by the small health app developer to the large medical institution.
13. C' represents the set of n_2 ciphertexts sent by the large medical institution to the small health app developer
14. $item_j$ represents the result obtained through a series of exchange and decryption operations, used to retrieve the filter to obtain the cardinality of intersection.

 doi:10.20944/preprints202405.0307.v1

15. sum represents the variable used to help the small health app developer obtain the cardinality of the intersection, where $sum - k$ represents the cardinality of the intersection.

6.2. Protocol Process

6.2.1. Preprocessing

1. Security parameter negotiation: Each role discusses the necessary security parameters, all parties share the large prime q used in the DH cryptographic algorithm. The small health app developer and the large medical institution negotiate to generate r_1 and the hash function H.
2. The small health app developer negotiates with the large medical institution to create an obfuscated dataset Y': This data set is completely useless data, which means that its elements cannot belong to either the small health app developer or the large medical institution collection..
3. Large medical institution generates a private key: The large medical institution generates its own private key α, for use in the Diffie-Hellman encryption algorithm.
4. Data scrambling: The small health app developer and the large medical institution each scramble their own datasets X and Y.
5. Small health app developer data preprocessing: The small health app developer calculates $h_j = H(y_j)$, generates n_2 random numbers β_j, and calculates $h_j \times r_1$.

6.2.2. Outsourcing

1. Large medical institution sends data to the cloud server: The large medical institution uses its private key α to perform the operation $CF.insert((H(x_i))^\alpha \times r_1^\alpha)$, creates a Cuckoo filter CF, and sends it to the cloud server.
2. Small health app developer sends data to the cloud server: The small health app developer sends the random numbers β_j and $h_j \times r_1$ to the cloud server. After receiving the data sent by the small health app developer, the cloud server calculates $C_j = r_1^{\beta_j} \times h_j^{\beta_j}$. At this point, the cloud server has saved the small health app developer's blinded data.

6.2.3. Cardinality Calculation

1. Cloud server sends data: The cloud server sends the blinded data C_j to the large medical institution.
2. Large medical institution processes data: Upon receiving C_j, the large medical institution uses its private key α to calculate $C'_j = C_j^\alpha$, and sends the result back to the cloud server.
3. Cloud server processes data: After receiving C'_j from the large medical institution, the cloud server calculates $item_j = C_j^{1/\beta_j}$ and uses the result to search CF. If $item_j$ exists in CF, then sum is incremented by 1 (initial value of sum is 0).
4. Obtaining the intersection cardinality: The small health app developer obtains the cardinality of the intersection by calculating $sum - k$, where k is the cardinality of the set Y'.

6.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, so $item_j = {C'_j}^{1/\beta_j} = {C_j}^{\alpha * 1/\beta_j} = \left({r_1}^{\beta_j} * h_j^{\beta_j}\right)^{1/\beta_j * \alpha} = {r_1}^\alpha * h_j^\alpha = {r_1}^\alpha * H(x_i)^\alpha = {r_1}^\alpha * H(y_j)^\alpha$.

Thus, through this scheme, the small health app developer can accurately obtain the cardinality of the intersection of both parties.

6.4. Security Analysis

In the design of this protocol, the primary security objective is to ensure that, even in a partially trusted cloud environment, neither the small health app developer's data nor the large medical institution's data can be accessed or inferred by unauthorized entities. Specifically, since other participating parties are unaware of the large medical institution's private key α, they cannot deduce the data

 doi:10.20944/preprints202405.0307.v1

held by the large medical institution. Similarly, since other parties do not know the small health app developer's private random number r_1, they cannot deduce the small health app developer's data.

However, this scheme has inherent security risks, primarily because it does not withstand collusion attacks. If the large medical institution and the cloud server collude, they can jointly deduce the small health app developer's data. This is possible because the cloud server possesses the blinded data $h_j \times r_1$, and if the large medical institution leaks the private key r_1 to the cloud server, then both the cloud server and the large medical institution could deduce the small health app developer's original data h_j. Collusion attacks are a security threat where two or more distinct entities (for example, users, systems, or service providers) secretly cooperate to undermine or circumvent security mechanisms and privacy measures. In cloud computing environments, cloud service providers and cloud users may collude to steal or infer other users' sensitive data stored on the cloud. In the medical scenario of this article, the intersection represents the patient's sensitive data, and leaking this information will cause very serious damage.

6.5. Experimental Analysis

6.5.1. Data Storage Volume

In the research of this paper, the experimental analysis of the unbalanced PSI-CA protocol based on single cloud assistance revealed a key issue: when the small health app developer needs to receive a Cuckoo filter from the large medical institution, this poses a significant challenge for receivers with limited storage capacity. This challenge is magnified when facing large datasets.

To understand this issue deeply, a series of experiments were conducted to measure the volume of Cuckoo filter needed by the small health app developer under different data sizes. The input data size for the experiments was provided by the large medical institution, reflecting the various data volumes that might be encountered in actual application scenarios. As shown in Table 3, the paper meticulously recorded the specific sizes of Cuckoo filter under different input data volumes, revealing the intrinsic relationship between data volume and filter size. Through experiments, it was discovered that as the data volume in the large medical institution increased, the storage burden on the small health app developer under the original protocol also increased accordingly, with the size of the Cuckoo filter directly impacted by the input data volume. Especially in the context of the large medical institution containing extensive patient data, this storage pressure is particularly evident.

Table 3. Size of Cuckoo Filter at Different Data Volumes

Data Set Count	Size of Cuckoo Filter (MB)
2^{15}	0.535
2^{17}	2.363
2^{20}	21.678
2^{22}	93.645
2^{23}	194.436
2^{24}	403.201
2^{27}	3571.206
2^{28}	7372.835
2^{29}	15206.421

Therefore, based on the above analysis and experimental results, it is clear that when the data volume in the large medical institution is excessively large, in other words, when the number of users reaches a certain level, the feasibility of a simple unassisted unbalanced PSI-CA protocol based on Cuckoo filter significantly decreases. This is because the unbalanced PSI-CA protocol based on Cuckoo filter requires small health app developers to directly receive and process massive Cuckoo filter, which poses a significant challenge for small health app developers with limited storage resources, particularly mobile devices. Small health app developer devices often do not have enough storage

 doi:10.20944/preprints202405.0307.v1

space to accommodate these large-volume filter data, let alone process these data to complete PSI-CA operations.

In this context, the introduction of a cloud server scheme shows its unique advantages. By transferring the storage of the filter to the cloud server, the burden on the small health app developer is greatly reduced. By this means, even in situations with a massive number of users and large data volumes, the scheme can still maintain efficient operations and ensure the smooth completion of PSI-CA operations.

In summary, through experimental and theoretical analysis, this section concludes that in scenarios with large-scale users and massive data volumes, the introduction of a cloud server scheme is more feasible and efficient than the unbalanced PSI-CA protocol based on Cuckoo filter.

6.5.2. Protocol Running Time

For this protocol, as shown in Table 4, the paper conducted experiments and recorded the running time of the protocol under various data volume combinations. Because preprocessing can be completed offline, the running time of the protocol refers to the total time of the outsourcing process and the intersection process. Table 4 also compares the running times of the unbalanced PSI-CA protocol based on Cuckoo filter and the unbalanced PSI-CA protocol based on single cloud assistance. Here, Protocol 1 refers to the unbalanced PSI-CA protocol based on Cuckoo filter, and Protocol 2 refers to unbalanced PSI-CA protocol based on single cloud assistance.

Table 4. Running Times of Protocol 1 and Protocol 2 Under Different Data Volume Combinations

Small Health App Developer Dataset Size	Large Medical Institution Dataset Size	Protocol 1 Running Time (seconds)	Protocol 2 Running Time (seconds)
2^{10}	2^{15}	0.1685	0.1658
2^{10}	2^{17}	0.1663	0.1693
2^{10}	2^{20}	0.1641	0.1658
2^{10}	2^{25}	0.1840	0.1731
2^{15}	2^{15}	5.2725	4.0627
2^{15}	2^{17}	5.5104	4.2464
2^{15}	2^{20}	5.2807	4.3202
2^{15}	2^{25}	5.5166	4.6118
2^{17}	2^{17}	21.1474	17.056
2^{17}	2^{20}	21.2865	16.731
2^{17}	2^{25}	21.9713	18.5417
2^{20}	2^{20}	171.5354	130.0498
2^{20}	2^{25}	186.9644	140.0193

From the experimental analysis, the following conclusions can be drawn: In cases of smaller data volumes, the performance differences between the two protocols are not significant. However, as the data volume increases, the running time differences between different protocols gradually become apparent. This is because, at certain specific levels, the proportion of communication time is relatively high when the data volume is small, significantly impacting the results. For larger data volumes, where computation time dominates, Protocol 2, by placing computational tasks on the more powerful cloud server, gradually widens the running time difference from Protocol 1. Overall, the use of cloud resources in the unbalanced PSI-CA protocol based on single cloud assistance significantly reduces running times, especially when dealing with large-scale datasets.

6.6. Summary of This Chapter

This chapter introduces an unbalanced PSI-CA protocol based on single cloud assistance, which utilizes cloud computing to reduce the computing and storage pressure of the small health app developer compared with previous protocols. Additionally, in the absence of collusion between the

 doi:10.20944/preprints202405.0307.v1

large medical institution and the cloud server, the protocol effectively protects data from unauthorized access, ensuring the confidentiality of the data and the privacy of the small health app developer, making it highly suitable for scenarios where the cloud server is fully trusted.

However, it cannot be denied that although this scheme significantly reduces the computational and storage burden on the small health app developer, its security against collusion attacks is insufficient. In the medical scenario of this article, the intersection represents the patient's sensitive data, and leaking this information will cause very serious damage.When the possibility of collusion between the cloud server and large medical institution cannot be completely ruled out, the protocol faces security risks and will require further security enhancement measures. Therefore, the next chapter will introduce a more secure solution to address the security deficiencies of the current scheme, ensuring the security and privacy of small health app developer data and large medical institution data in environments where not all parties are fully trustworthy. In other words, the new scheme can resist collusion attacks.

7. Unbalanced PSI-CA Protocol Based on Dual cloud Assistance

The previous single-server solution, which efficiently delegated computationally intensive encryption operations such as exponentiation and storage-intensive Cuckoo filter to the cloud server, has indeed alleviated the computational and storage burdens on the small health app developer to a certain extent. This is particularly advantageous for small health app developers with limited computing and storage capabilities, allowing them to operate beyond their hardware constraints. However, security analysis reveals that the unbalanced PSI-CA protocol based on single cloud assistance has inherent security risks, specifically when collusion between the cloud server and large medical institution is possible, thus compromising its adequacy in protecting small health app developer data privacy.

As shown in Figure 3, to preserve the advantages of the previous scheme—namely reducing computational and storage pressures on the small health app developer—while addressing these security issues, this chapter proposes a new solution. This design aims to enhance the security during data processing, especially against potential collusion attacks.

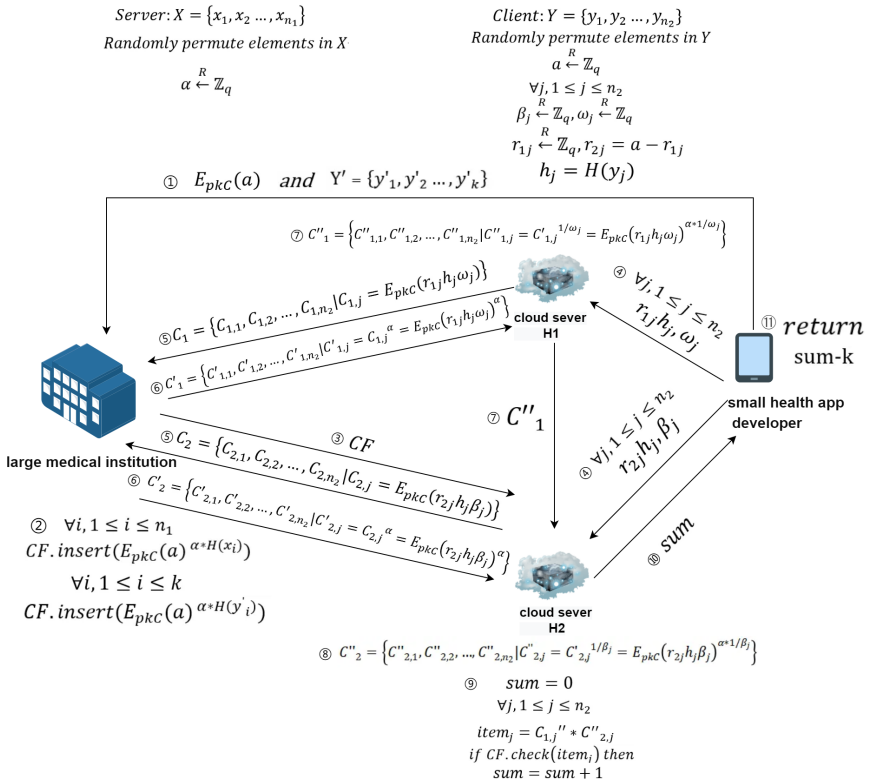

Figure 3. Unbalanced PSI-CA Protocol Based on Dual cloud Assistance

7.1. Definition of Main Participants and Related Symbols

1. large medical institution represents the party with a larger dataset and greater computational and storage capabilities.
2. small health app developer represents the party with a smaller dataset and lesser computational and storage capabilities.
3. cloud server H_1: Acts as an auxiliary server for the small health app developer, handling the majority of computation and storage pressures.
4. cloud server H_2: Another auxiliary server handling substantial computational and storage demands.
5. X and Y: Represent the dataset of the large medical institution and the small health app developer, respectively.
6. $Y^{'}$ represents the obfuscated dataset sent by the small health app developer to the large medical institution, used to confuse the cloud server and prevent it from obtaining the accurate cardinality of the intersection.k represents the cardinality of the set $Y^{'}$.
7. α: Represents the private key of the large medical institution used in the Diffie-Hellman encryption algorithm.
8. H: The hash function agreed upon by the small health app developer and the large medical institution for use.
9. CF: Represents the Cuckoo Filter, where $CF.insert$ denotes the operation to add elements, and $CF.check$ checks for the presence of specific elements.
10. ω_j: Random exponentials generated by the small health app developer for cloud server H_1, β_j for cloud server H_2.
11. a: A secret value held by the small health app developer.
12. $r_{1,j}$: Random numbers used by the small health app developer for sending obfuscated data to cloud server H_1, and $r_{2,j}$ for H_2 where $r_{1,j} + r_{2,j} = a$.
13. C_1: The ciphertext collection sent from cloud server H_1 to the large medical institution, and C_2 from H_2; $C_{1,j}$ and $C_{2,j}$ are specific elements within these collections.
14. C_1' and C_2': Processed ciphertext collections returned to H_1 and H_2 from the large medical institution; $C_{1,j}'$ and $C_{2,j}'$ are specific elements within these collections.
15. C_1'' and C_2'': Final processed ciphertext collections at H_1 and H_2 after receiving data from the large medical institution; $C_{1,j}''$ and $C_{2,j}''$ are specific elements within these collections.
16. $item_j$: Represents the result of multiplying $C_{1,j}''$ and $C_{2,j}''$ used to query the filter.
17. sum represents the variable used to help the small health app developer obtain the cardinality of the intersection, where $sum - k$ represents the cardinality of the intersection.

7.2. Protocol Process

7.2.1. Preprocessing

1. Discuss security parameters: Each party discusses the necessary security parameters—the large prime q used in DH encryption and the small health app developer's public key pk_c required for the Paillier encryption system. The small health app developer and the large medical institution negotiate the creation of hash function H.
2. The small health app developer negotiates with the large medical institution to create an obfuscated dataset $Y^{'}$: This data set is completely useless data, which means that its elements cannot belong to either the small health app developer or the large medical institution collection.
3. small health app developer sends $E_{pk_c}(a)$: The small health app developer generates its private secret number a and sends $E_{pk_c}(a)$ to the large medical institution.
4. large medical institution generates private key: The large medical institution creates its private key α, used for the DH encryption algorithm.
5. Data scrambling: The small health app developer and the large medical institution each shuffle their respective datasets.

 doi:10.20944/preprints202405.0307.v1

6. small health app developer calculates hashes and generates random numbers: The small health app developer computes $h_j = H(y_j)$ and generates n_2 random numbers β_j, ω_j, r_{1j}, r_{2j}, and computes $r_{1j}h_j$, $r_{2j}h_j$ where $r_{1j} + r_{2j} = a$.

7.2.2. Outsourcing

1. small health app developer sends data to cloud servers: The small health app developer sends $r_{1j}h_j$, ω_j to cloud server H_1, and $r_{2j}h_j$, β_j to cloud server H_2. H_1 computes $C_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)$, and H_2 computes $C_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)$. At this point, H_1 and H_2 hold the small health app developer's obfuscated data.
2. large medical institution sends data to cloud servers: Using $E_{pk_c}(a)$, the large medical institution performs the filter insertion operation CF.insert$\left(E_{pk_c}(a)^{\alpha * H(x_i)}\right)$ to generate a Cuckoo filter and sends it to cloud server H_2. H_2 stores the filter sent by the large medical institution.

7.2.3. Intersection

1. H_1 and H_2 send data: H_1 and H_2 each send their respective collections C_1 and C_2 to the large medical institution.
2. large medical institution processes data: Upon receiving the data, the large medical institution uses its private key α to compute $C'_{1,j} = C^{\alpha}_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha}$ and sends the results back to H_1. It also processes $C'_{2,j} = C^{\alpha}_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha}$ and sends the results back to H_2.
3. H_1 processes data: After receiving data from the large medical institution, H_1 uses the random number ω_j to calculate $C''_{1,j} = C'^{1/\omega_j}_{1,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha * 1/\omega_j}$ and sends the results to H_2.
4. H_2 processes data: Upon receiving data from H_1 and the large medical institution, H_2 calculates $C''_{2,j} = C'^{1/\beta_j}_{2,j} = E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha * 1/\beta_j}$. H_2 checks if $item_j = C''_{1,j} * C''_{2,j}$ exists in CF. If $item_j$ exists in CF, then sum is incremented by 1 (initial value of sum is 0).
5. Obtaining the intersection cardinality: The small health app developer obtains the cardinality of the intersection by calculating $sum - k$, where k is the cardinality of the set Y'.

7.3. Correctness Analysis

If $x_i = y_j$, then $H(x_i) = H(y_j)$, which implies that $C''_{1,j} * C''_{2,j} = C'^{1/\omega_j}_{1,j} * C'^{1/\beta_j}_{2,j} = E_{pk_c}(r_{1j}h_j\omega_j)^{\alpha * 1/\omega_j} * E_{pk_c}(r_{2j}h_j\beta_j)^{\alpha * 1/\beta_j} = E_{pk_c}(r_{1j}h_j\alpha) * E_{pk_c}(r_{2j}h_j\alpha) = E_{pk_c}[\alpha h_j(r_{1j} + r_{2j})] = E_{pk_c}(\alpha h_j a) = E_{pk_c}(a)^{\{\alpha * H(x_i)\}}$.

Thus, through this scheme, the small health app developer can accurately obtain the cardinality of the intersection of both parties.

7.4. Security Analysis

Firstly, we consider the security ofsmall health app developer's data in the set.In considering security against collusion attacks, it is generally assumed that there is an adversary who possesses the perspective and information of all participating parties except for the protected entity. This means the adversary can access, control, or receive information and resources from all participants except for the small health app developer. In this scenario, the adversary attempts to compromise the system's security or privacy by aggregating these insights, such as revealing sensitive data of the small health app developer. If, in this context, the adversary still cannot learn or infer the small health app developer's data, then it is proven that the data and privacy of the small health app developer are sufficiently secured against collusion attacks.

This section defines a game where the security objective is to maintain confidentiality of the data within the set under semi-honest and collusion conditions. The game for securing the small health app developer's data set is as follows:

1. The small health app developer runs the preprocessing algorithm, sharing the cryptographic hash function H and the large prime q used in the protocol with the adversary.

 doi:10.20944/preprints202405.0307.v1

2. The small health app developer simulates the outsourcing algorithm and sends their (encrypted) input to the adversary.
3. The small health app developer and the adversary simulate the intersection algorithm and discard any output.
4. The adversary is asked to output a guess $\hat{y}$ of the small health app developer's input y.

The game is analogized to a deterministic one-way function, such as a public key encryption scheme. Let S be the simulated messages of the small health app developer during the game. Let a one-way function adversary A' be given the information (public key) pk and function (ciphertext) c (encrypted y). The advantage of the adversary Adv_A is defined as the difference between the successful guesses of A and A'. If this advantage is negligible in the security parameter λ, then the outsourced private set intersection is considered secure. That is, let $Adv_A = \Pr[A(S) = y] - \Pr[A'(pk, c) = y]$. If $Adv_A < \frac{1}{\text{poly}(\lambda)}$, then the protocol is said to be secure.

Specifically, after the steps mentioned above, H_1, H_2, and the large medical institution have a complete view of the process. However, under the two-server architecture, as illustrated in Figure 3:

1. In step four of Figure 3, since r_{1j} and r_{2j} are unknown to the adversary, h_j cannot be derived. The adversary can only attempt exhaustive guessing, thus making Adv_A negligible.
2. In subsequent steps, as A does not know the small health app developer's private key for the Paillier encryption system, it is impractical to decrypt the ciphertexts, making it even more challenging to derive h_j. For instance, $item_j = C''_{1,j} * C''_{2,j} = {C'_{1,j}}^{1/\omega_j} * {C'_{2,j}}^{1/\beta_j} = E_{pk_c}(r_{1j} h_j \omega_j)^{\alpha * 1/\omega_j} * E_{pk_c}(r_{2j} h_j \beta_j)^{\alpha * 1/\beta_j}$, and since the private key used in Paillier's system by the small health app developer is unknown, decrypting this compound is complex and hence h_j remains secure.

From the analysis above, it is evident that the advantage of Adv_A is negligible. Therefore, if both cloud servers collude with the large medical institution, they cannot deduce the small health app developer's original data.

Next, consider the security of the data in the large medical institution's set. Obviously, apart from the large medical institution itself, none of the parties know the large medical institution's private key α, hence even if both cloud servers colluded with the small health app developer, they cannot derive the original data from CF.

It is particularly noted that due to the prevalence of attacks on hash functions, further security enhancements are recommended by protecting the hashed data as the raw data.

In addition, due to the existence of obfuscated dataset Y', two cloud servers cannot know the set cardinality of the large medical institution and the small health app developer.

In conclusion, the dual-server scheme successfully resists collusion attacks under semi-honest conditions. By thoroughly integrating considerations for security and privacy into the protocol design, both the small health app developer's and the large medical institution's data are assured of robust protection. This solution not only provides an effective mechanism for private set intersection but also demonstrates resilience against potential collusion threats.

7.5. Experimental Analysis

7.5.1. Data Computation Volume

When evaluating the performance of these protocols, the computational load borne by the small health app developer is undoubtedly a critical factor. Since all three protocols have been introduced, this section specifically focuses on the computational volume of the small health app developer to accurately gauge and compare the efficiency of the three distinct protocols in operation. Specifically, this section will conduct a detailed analysis and comparison of the main computational tasks that the small health app developer must execute across these protocols, to fully assess each protocol's demand on the small health app developer's computational resources. This analysis will primarily focus on the types of operations involved, aiming to clarify which protocol demonstrates relative

 doi:10.20944/preprints202405.0307.v1

advantages in reducing the small health app developer's computational burden, thus providing a solid basis for selecting the most appropriate protocol. Below is an analysis of the main types of operations involved in each protocol, focusing primarily on the outsourcing and intersection processes, as the preprocessing can be completed offline. .

1. **unbalanced PSI-CA protocol based on Cuckoo filter:** Two rounds of modular exponentiation operations and filter retrieval.
2. **unbalanced PSI-CA protocol based on single cloud assistance**: A single round of multiplication operations .
3. **unbalanced PSI-CA protocol based on dual cloud assistance**: Two rounds of multiplication operations .

An analysis of the single-instance time consumption for these four operations offers a practical insight into the computational volume differences:

1. **Modular Exponentiation Operation**: Representing computation-intensive operations, modular exponentiation becomes particularly time-consuming. On a standard hardware setup, the time required for a single modular exponentiation operation depends primarily on the size of the numbers involved and the efficiency of the algorithm.
2. **Multiplication Operation**: Compared to modular exponentiation, multiplication operations execute much faster on modern computing systems, even when involving large numbers. Therefore, whether it's a single round of multiplication in the single-cloud protocol or two rounds in the dual-cloud protocol, the processing times are relatively short.
3. **Cuckoo Filter Retrieval**: Although relatively quick, the retrieval operation for a Cuckoo filter involves memory access, which may make it slightly slower than simple arithmetic operations. The exact time required for this operation depends on the size of the filter and the efficiency of the implementation.

After a detailed analysis and comparison, this section has conducted a thorough exploration of the key computational tasks executed by the small health app developer across the three different protocols. These tasks include modular exponentiation, multiplication operations, and Cuckoo filter retrieval By assessing these types of computations and their specific time consumptions, the following conclusions can be drawn:

1. **Unbalanced PSI-CA Protocol Based On Cuckoo Filter**: Primarily relies on two rounds of modular exponentiation and, which are computation-intensive, especially when dealing with large numbers, making it the most time-consuming of all the operations reviewed.Additionally, the filter retrieval operation is also involved.
2. **Unbalanced PSI-CA Protocol Based On Single Cloud Assistance** : By executing a single round of multiplication, it significantly alleviates the computational burden on the small health app developer. Multiplication operations, even for large numbers, can be done quickly.
3. **Unbalanced PSI-CA Protocol Based On Dual cloud Assistance**: Includes two rounds of multiplication operations, also aiming to distribute the computational pressure on the small health app developer. Although it involves two rounds of multiplication, due to the inherent efficiency of the operation, the total processing time remains within an acceptable range.

Through the meticulous assessment of each protocol's computational types and their time consumptions, it is evident that both the unbalanced PSI-CA protocol based on single cloud assistance and unbalanced PSI-CA protocol based on dual cloud assistance exhibit excellent performance in reducing the small health app developer's computational burden, particularly in the efficient execution of multiplication operations. In contrast, the unbalanced PSI-CA protocol based on Cuckoo filter, while potentially offering stronger security provisions, shows some deficiencies in efficiency and timeliness. Therefore, when choosing an appropriate protocol, a balance should be struck based on actual performance requirements and security needs.

 doi:10.20944/preprints202405.0307.v1

7.5.2. Protocol Running Time

After introducing all three protocols, this section primarily discusses the running times of the protocols. The running time of a protocol is an important benchmark for evaluation in this paper because it directly reflects the protocol's efficiency in practical operations. The factors affecting the running time of the protocol include computational time and communication time. As shown in Table 5, experiments were conducted to record the running times of the protocols under various data volume combinations.Table 5 also places the running times of the unbalanced PSI -CA protocol based on Cuckoo filter, unbalanced PSI-CA protocol based on single cloud assistance, and unbalanced PSI-CA protocol based on dual cloud assistance side by side for comparative analysis. Here, Protocol I refers to the unbalanced PSI-CA protocol based on Cuckoo filter, Protocol II refers to the unbalanced PS-CAI protocol based on single cloud assistance, and Protocol III refers to the unbalanced PSI-CA protocol based on dual cloud assistance.

Table 5. Running times of the three protocols under different data volume combinations

Data Volume	Protocol I Running Time (s)	Protocol II Running Time (s)	Protocol III Running Time (s)
$2^{10}\|2^{15}$	0.1539	0.1543	0.1612
$2^{10}\|2^{17}$	0.1569	0.1573	0.1742
$2^{10}\|2^{20}$	0.1616	0.1611	0.1736
$2^{10}\|2^{25}$	0.1693	0.1683	0.1868
$2^{15}\|2^{15}$	4.9239	3.8223	4.3904
$2^{15}\|2^{17}$	5.0232	3.9145	4.9128
$2^{15}\|2^{20}$	5.1709	4.0267	4.6099
$2^{15}\|2^{25}$	5.4172	4.2233	4.8281
$2^{17}\|2^{17}$	20.0930	15.6768	18.2058
$2^{17}\|2^{20}$	20.6841	16.1281	20.2155
$2^{17}\|2^{25}$	21.6690	16.8939	20.0528
$2^{20}\|2^{20}$	165.4731	129.0516	148.7145
$2^{20}\|2^{25}$	173.3531	135.2534	165.9464

It is noteworthy that the preprocessing stages of all three protocols can be completed offline, meaning they do not directly contribute to online operation delays. Therefore, the recorded running times in this paper refer to the total time of all processes excluding preprocessing. Specifically, in Protocol I, this primarily refers to the total duration of the intersection process; in Protocols II and III, it refers to the total duration of both the outsourcing and intersection processes.

Through experimental analysis, the paper draws the following conclusions: At smaller data volumes, the performance differences between the three protocols are not significant. However, as the data volume increases, the differences in running times between the protocols become apparent. Generally, the unbalanced PSI-CA protocol based on Cuckoo filter tends to have the longest running time, while the unbalanced PSI-CA protocol based on single cloud assistance has the shortest running time, and the performance of the unbalanced PSI-CA protocol based on dual cloud assistance is in the middle. This phenomenon can be explained by the complexity of data handling and the differences in communication overhead among the protocols. The unbalanced PSI-CA protocol based on Cuckoo filter, due to its direct and unoptimized calculations, is less efficient when handling large volumes of data. Nevertheless, at very small data volumes, where the proportion of communication time is relatively high, the impact of data transmission costs on total running time becomes significant. In such cases, the unbalanced PSI-CA protocol based on Cuckoo filter does not necessarily appear inefficient because other protocols might be even less efficient in data transmission. Especially in environments with poor network conditions or limited data transfer rates, the lower communication demands of the unbalanced PSI-CA protocol based on Cuckoo filter might, in some cases, lead to better performance.

Moreover, the running times of the unbalanced PSI-CA protocol based on single cloud assistance and the unbalanced PSI-CA protocol based on dual cloud assistance are significantly reduced through

 doi:10.20944/preprints202405.0307.v1

distributed computing and the use of cloud resources, especially when dealing with large-scale datasets. In summary, choosing the appropriate protocol requires a comprehensive consideration of factors such as data volume, computational resources, and network environment. In practical applications, understanding the performance characteristics and suitable scenarios of each protocol is crucial for optimizing data processing workflows and enhancing efficiency.

7.6. Summary of This Chapter

The protocol leverages the computational and storage resources of two cloud servers, significantly reducing the burden on the small health app developer by lowering its computational and storage requirements and enhancing the system's efficiency and availability. Through distributed computing and security measures such as homomorphic encryption, it ensures the privacy of data during transmission and processing, adequately protecting sensitive information of both the small health app developer and the large medical institution. This solution not only improves the operational efficiency of devices with limited resources but also effectively prevents collusion attacks. Consequently, the unbalanced PSI-CA protocol based on dual cloud assistance excels in private set intersection operations, ,demonstrating both high efficiency and security.

7.7. Extensions

To enhance the practicality of the scheme, this section will explore two key aspects from an engineering practice perspective: the design of the PSI-CA network and the design of the data update mechanism.

First, the design of the PSI-CA network focuses on building an efficient, secure, and scalable network architecture to support large-scale PSI-CA computations.

Second, the design of the data update mechanism involves how to update the data sets stored on the cloud servers without interrupting the service. This is particularly crucial for PSI-CA computation scenarios that require frequent data updates.

7.7.1. PSI-CA Network

As previously described, the small health app developer delegates PSI-CA computations to two cloud servers. In practice, a vast network of cloud servers can be built to support this delegation. The basic system description is as follows.

- Access and Authentication of Cloud Servers: Any server can apply to become a cloud server, also known as a server assistant. These servers must undergo a series of certification processes (including hardware performance verification, security vulnerability scanning, and compliance checks) to ensure they meet security and performance standards. Servers that pass the certification but later violate regulations will be blacklisted and removed. The system maintains platform security and trust through mechanisms such as regular security scans and real-time monitoring, with any violations leading to immediate removal and further investigation of the server.
- Mechanism for Selecting Server Assistants: When needing to perform PSI-CA, small health app developers choose two cloud servers based on their performance (such as processing power, storage capacity, and network bandwidth), stability, security capabilities, and compliance with regulations, among other hard and soft factors. Cloud servers with high availability promises are preferred to minimize the risk of failures.
- Execution Mechanism for PSI-CA Operations: The PSI-CA network supports small health app developer flexibility and system scalability; small health app developers can execute PSI-CA on different large medical institutions by merely changing $E_{pk}(a)$ and obfuscated dataset $Y^{'}$, without needing to redesign the entire system. This design enhances small health app developer flexibility and the system's efficiency, reliability, and security.

This system design not only achieves the delegation of PSI-CA computations but also introduces multiple cloud servers into the network, thereby enhancing the system's flexibility and stability.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 7 May 2024 doi:10.20944/preprints202405.0307.v1

Additionally, by implementing authentication and maintaining a blacklist for cloud servers, the system can better guarantee the credibility of the cloud servers, enhancing overall security. This flexible yet secure system design provides small health app developers with more options and makes PSI-CA operations more adaptable to various practical requirements.

In summary, under the existing framework, small health app developers can delegate computing and storage tasks to different cloud servers and perform PSI-CA operations on various large medical institutions by using different random numbers , $E_{pk}(a)$ and and obfuscated dataset Y'. This method allows small health app developers to more flexibly use multiple resource nodes and optimize task distribution, thereby further enhancing the overall performance and security of the privacy protection scheme.

7.7.2. Data Updates

To further enhance the practicality of the scheme, this paper also designs a data update mode compatible with the scheme, making the overall scheme more practical and reliable.

- **Data Updates on the Large Medical Institution's Side:**

  As shown in Figure 4, the update details of the large medical institution are as follows:

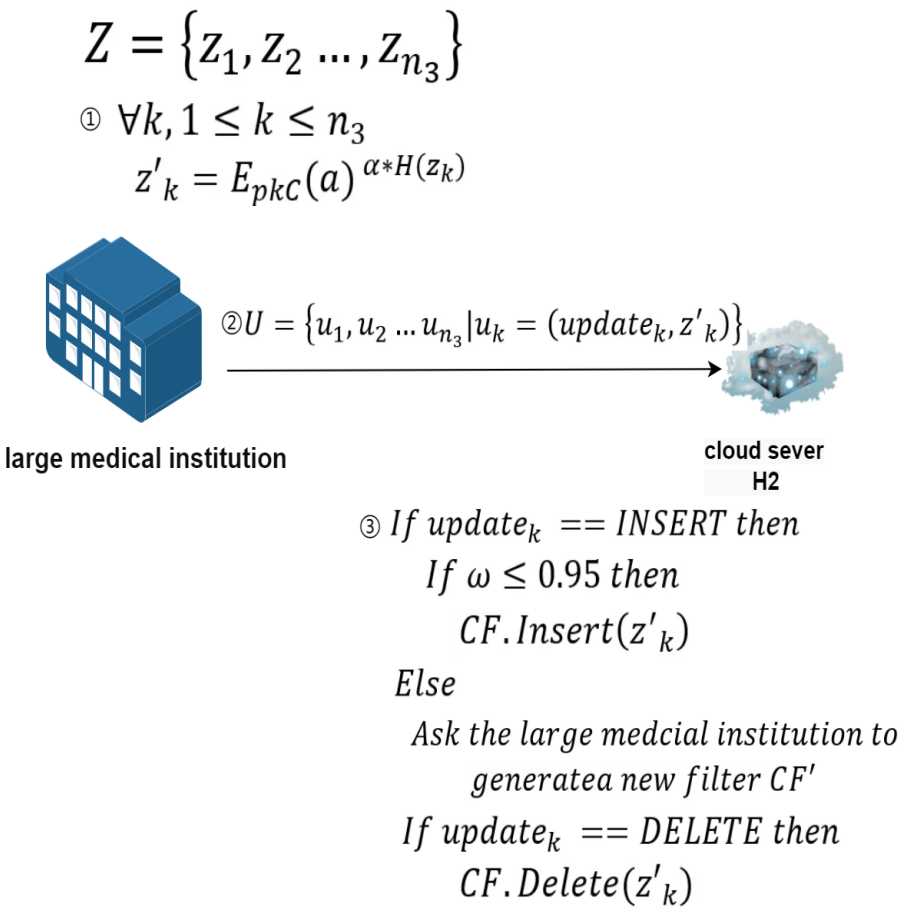

Figure 4. Data Updates on the Large Medical Institution's Side

Definition of main participants and related symbols:

1. Large medical institution: Represents the large medical institution that wants to encrypt and upload updated data to cloud server H_2.
2. Cloud server H_2: Represents the cloud-assisted server H_2 that assists the large medical institution in completing update operations.

 doi:10.20944/preprints202405.0307.v1

3. Z represents the set of data to be updated, z_k represents the k-th element of Z.
4. ω represents the load factor of the filter.
5. z'_k represents the data z_k after encryption processing.
6. update_k represents the operation index, used to determine whether the update operation is an insertion or deletion.
7. U represents the set of data sent by the large medical institution to the cloud-assisted server H_2, u_k represents the k-th element of U.

Update process:

1. The large medical institution has a set of elements Z it wants to insert or delete. These elements are blinded before being sent to cloud server H_2. Specifically, $z'_k = E_{pk_c}(a)^{\alpha * H(z_k)}$.
2. In addition to sending the blinded elements, the large medical institution also sends an identifier variable update_k to inform the small health app developer whether the operation is an insertion or a deletion.
3. During an insertion operation, H_2 first checks whether the current filter's load factor ω exceeds 0.95.
4. If the load factor ω is greater than 0.95, then H_2 must request the large medical institution to generate a new filter using all elements to maintain high spatial and lookup efficiency of the filter.
5. If the load factor ω is less than or equal to 0.95, then H_2 can directly insert the element into the current filter CF.
6. In a deletion operation, H_2 removes the specified element from the filter CF, a process that does not require generating a new filter.

- **Data Updates on the Small Health App Developer's Side:**

As shown in Figure 5, the update details of the small health app developer are as follows:

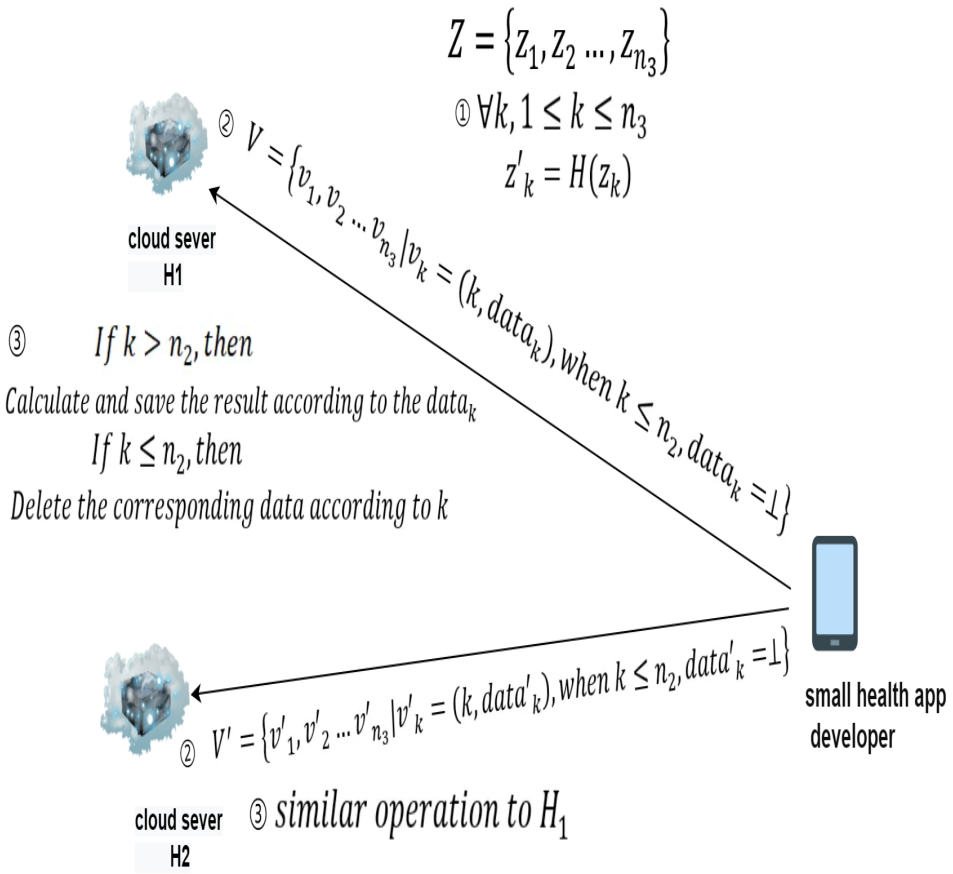

Figure 5. Data Updates on the Small Health App Developer's Side

 doi:10.20944/preprints202405.0307.v1

Definition of main participants and related symbols:

1. Small health app developer: Represents the small health app developer who wants to perform data updates.
2. Cloud server H_1: Represents the cloud-assisted server H_1 that assists the small health app developer in completing update operations.
3. Cloud server H_2 Represents the cloud-assisted server H_2 that assists the small health app developer in completing update operations.
4. Z represents the set of data to be updated, z_k represents the k-th element of Z.
5. z'_k represents the data after being processed by the hash function H.
6. k represents the data index, used to determine the type of update, either insertion or deletion, and to retrieve the updated data based on the index.
7. When adding data, date_k represents the data processed through the dual-cloud scheme and sent to the two cloud-assisted servers. When deleting, date_k is null.
8. V represents the set of data sent by the small health app developer to the cloud-assisted server H_1, v_k represents the k-th element of V.
9. V' represents the set of data sent by the small health app developer to the cloud-assisted server H_2, v'_k represents the k-th element of V'.

Update process:

1. The small health app developer has a set of elements Z it wants to insert or delete. In both cases, the small health app developer blinds each element and sends them to H_1 and H_2 respectively.
2. The small health app developer sends a data index K to inform the cloud servers about the type of update, whether it is an insertion or a deletion. If the index is less than n_2, it indicates a deletion operation. In this case, data_k is null, and H_1 and H_2 delete the corresponding data based on the index.
3. If the index is greater than n_2, it indicates an addition operation, and the corresponding calculation results and index are saved.
4. After completing a batch of deletion and addition operations, the relative order of the indices also needs to be adjusted. The update process is illustrated in Figure 5.

8. Conclusions and Future Work

8.1. Work Summary

Privacy computing is a technology framework aimed at protecting individual privacy during the process of data use and sharing. It ensures that data can still be effectively utilized without disclosing specific content through various algorithms and protocols. Among many applications of privacy computing, Privacy Set Intersection (PSI-CA) is a common requirement, which allows two or more parties to compute the cardinality of the intersection without revealing their private data.

Traditional PSI-CA protocols are primarily designed for cases where data sets are relatively balanced in size, which often does not apply in real scenarios. In many practical situations, the size disparity between participants' data sets is significant, necessitating the use of unbalanced PSI-CA protocols. These protocols are specifically designed to handle such disparities, optimizing computational efficiency and privacy protection to cater to a wider range of practical needs. By adopting unbalanced PSI-CA protocols, not only is the processing efficiency improved, but more precise control over data protection is also offered, thus finding broader application in various data-sensitive industries. This paper proposes three protocols: the unbalanced PSI-CA protocol based on Cuckoo filter, the unbalanced PSI-CA protocol based on single cloud assistance, and the unbalanced PSI-CA protocol based on dual cloud assistance. Here, the unbalanced PSI-CA protocol based on Cuckoo filter addresses performance issues of traditional PSI-CA protocols in handling unbalanced data sets. On this basis, the unbalanced PSI-CA protocol based on single cloud assistance transfers most of the computational and storage burdens from the small health app developer to the cloud, enhancing practicality. Faced with

 doi:10.20944/preprints202405.0307.v1

the possibility of collusion attacks, the unbalanced PSI-CA protocol based on dual cloud assistance employs security mechanisms such as homomorphic encryption to effectively resist these attacks. The main contributions of this paper are summarized as follows:

1. Addressing the shortcomings of traditional PSI-CA protocols when dealing with significant data size disparities among participants, this paper proposes the first protocol, namely the unbalanced PSI-CA protocol based on Cuckoo filter.
2. Given the complexities of cryptographic operations and storage demands of the small health app developer in the unbalanced PSI-CA protocol based on Cuckoo filter, this paper introduces a unbalanced PSI-CA protocol based on single cloud assistance. This protocol effectively transfers the majority of computational and storage burdens from the small health app developer to the cloud.
3. In response to potential collusion between the cloud and large medical institution in the unbalanced PSI-CA protocol based on single cloud assistance, this paper proposes a unbalanced PSI-CA protocol based on dual cloud assistance with security mechanisms like homomorphic encryption, which effectively prevents collusion attacks while offloading computational and storage burdens.
4. In view of the practical problems of the unbalanced PSI-CA protocol based on dual cloud assistance, this paper also designs a PSI-CA network and a data update mode tailored for the unbalanced PSI-CA protocol based on dual cloud assistance.

8.2. Protocol Summary

As shown in Table 6, this section provides a comprehensive summary and recommendations for the three protocols discussed in this paper. The unbalanced PSI-CA protocol based on Cuckoo filter offers high security but involves significant computational and storage demands, making it suitable for clients with strong computational and storage resources. The unbalanced PSI-CA protocol based on single cloud assistance, while being the fastest and offloading computational burdens to the cloud, poses security risks as it cannot withstand collusion attacks, making it appropriate for scenarios where the cloud is fully trusted. The unbalanced PSI-CA protocol based on dual cloud assistance offers an ideal balance of runtime, security, and efficiency, making it the most versatile and practical option.

Table 6. Overall Performance Summary of the Three Protocols

Protocol	**Security**	**Client Storage & Computational Burden**	**Runtime**
Unbalanced PSI-CA Protocol based on Cuckoo Filter	High Security (No collusion attacks)	Requires storing Cuckoo filter and intensive computation	Longest
Unbalanced PSI-CA Protocol based on Single Cloud Assistance	Security Risks (Cannot resist collusion attacks)	Shifted to cloud server	Fastest
Unbalanced PSI-CA Protocol based on Dual Cloud Assistance	High Security (Can resist collusion attacks)	Shifted to cloud server	Moderate

8.3. Future Outlook

Although the protocols proposed in this document are applicable in most scenarios, there are still several aspects that could be optimized for future development:

 doi:10.20944/preprints202405.0307.v1

1. All protocols are designed for two-party unbalanced PSI-CA. Extending these protocols to multi-party scenarios is an important future direction, given the practical needs for multi-party computations.
2. The protocols are developed under a semi-honest security model. Extending their robustness to malicious models, where adversaries may actively attempt to undermine the protocols, represents a crucial area for further research.
3. The current protocols are focused exclusively on PSI-CA. In practical applications, there may be a need to carry out other types of computations, such as PSI-SUM, etc. Expanding the protocols to support a variety of computational types is another significant direction for future work.

References

1. Bald, P.; Baronio, R.; Cristofaro, E.; Gasti, P.; Tsudik, G. Efficient and secure testing of fully-sequenced human genomes. *Biological Sciences Initiative* **2000**, *470*, 7–10.
2. Chen, H.; Laine, K.; Rindal, P. Fast private set intersection from homomorphic encryption. In Proceedings of the Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1243–1255.
3. Nagaraja, S.; Mittal, P.; Hong, C.Y.; Caesar, M.; Borisov, N. {BotGrep}: Finding {P2P} Bots with Structured Graph Analysis. In Proceedings of the 19th USENIX Security Symposium (USENIX Security 10), 2010.
4. Li, W.; Liu, J.; Zhang, L.; Wang, Q.; He, C. A Survey on Set Intersection Computation for Privacy Protection. *Journal of Computer Research and Development* **2022**, *59*, 1782–1799.
5. Meadows, C. A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party. In Proceedings of the Proc. of the 7th IEEE Symposium on Security and Privacy, Los Alamitos, CA, 1986; pp. 134–134.
6. Huberman, B.; Franklin, M.; Hogg, T. Enhancing Privacy and Trust in Electronic Communities. In Proceedings of the Proc. of the 1st ACM Conference on Electronic Commerce, New York, 1999; pp. 78–86.
7. DeCristofaro, E.; Tsudik, G. Experimenting with Fast Private Set Intersection. In Proceedings of the Proc. of Int. Conf. on Trust and Trustworthy Computing, Berlin, 2012; pp. 55–73.
8. Pinkas, B.; Schneider, T.; Zohner, M. Faster Private Set Intersection Based on OT Extension. In Proceedings of the Proc. of the 23rd USENIX Security Symposium, Berkeley, CA, 2014; pp. 797–812.
9. Freedman, M.; Nissim, K.; Pinkas, B. Efficient Private Matching and Set Intersection. In Proceedings of the Proc. of the 23rd Int. Conf. on the Theory and Applications of Cryptographic Techniques, Berlin, 2004. Accessed: 2020-10-16.
10. Freedman, M.J.; Hazay, C.; Nissim, K.; et al.. Efficient Set Intersection with Simulation-Based Security. *Journal of Cryptology* **2016**, *29*, 115–155.
11. Abadi, A.; Terzis, S.; Dong, C. O-PSI: Delegated Private Set Intersection on Outsourced Datasets. In Proceedings of the Proc of the 27th IFIP International Information Security and Privacy Conference, Berlin, 2015; pp. 3–17.
12. Kissner, L.; Song, D. Privacy-Preserving Set Operations. In Proceedings of the Proc of the 25th Annual International Cryptology Conference, Berlin, 2005; pp. 241–257.
13. Jarecki, S.; Liu, X. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In Proceedings of the LNCS 5444: Proc of the 6th Theory of Cryptography Conference, Berlin, 2009; pp. 577–594.
14. Hazay, C.; Venkitasubramaniam, M. Scalable Multi-party Private Set-Intersection. In Proceedings of the Proc of the 20th IACR International Workshop on Public Key Cryptography, Berlin, 2017; pp. 175–203.
15. Dou, J.; Liu, X.; Wang, W.; et al.. Efficient and Secure Calculation of Two-Party Sets in the Field of Rational Numbers. *Chinese Journal of Computers* **2020**, *43*, 1397–1413.
16. Damgård, I.; Pastro, V.; Smart, N.; et al. Multiparty Computation from Somewhat Homomorphic Encryption. In Proceedings of the Proceedings of the 32nd Annual Cryptology Conference; Name, E., Ed., Berlin, 2012; Lecture Notes in Computer Science, pp. 643–662.
17. Yao, A.C. Protocols for Secure Computations. In Proceedings of the Proc of the 23rd Annual Symposium on Foundations of Computer Science (SFCS 1982), Piscataway, NJ, 1982; pp. 160–164.

18. Micali, S.; Goldreich, O.; Wigderson, A. How to Play Any Mental Game. In Proceedings of the Proc of the 19th ACM Symposium on Theory of Computing, New York, 1987; pp. 218–229.
19. Pinkas, B.; Schneider, T.; Segev, G.; et al.. Phasing: Privateset intersectionusing permutation-basedhashing. In Proceedings of the Proceedings of the 24th USENIX Security Symposium, USENIX Association, Berkeley, CA, 2015; pp. 515–530.
20. Pinkas, B.; Schneider, T.; Weinert, C.; et al.. Efficient circuit-based PSI via cuckoo hashing. In Proceedings of the Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, 2018; pp. 125–157.
21. Pinkas, B.; Schneider, T.; Tkachenko, O.; et al.. Efficient circuit-based PSI with linear communication. In Proceedings of the Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, Berlin, 2019; pp. 122–153.
22. Huang, Y.; Evans, D.; Katz, J. Private Set Intersection: Are Garbled Circuits Better Than Custom Protocols? In Proceedings of the Proc of the 19th Network and Distributed System Security Symposium, Reston, VA, 2012. Accessed: 2020-10-21.
23. Naor, M.; Pinkas, B. Efficient oblivious transfer protocols. In Proceedings of the SODA, 2001, Vol. 1, pp. 448–457.
24. Dong, C.; Chen, L.; Wen, Z. When private set intersection meets big data: an efficient and scalable protocol. In Proceedings of the Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, 2013, pp. 789–800.
25. Rindal, P.; Rosulek, M. Improved private set intersection against malicious adversaries. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer International Publishing, 2017, pp. 235–259.
26. Zhang, E.; Liu, F.H.; Lai, Q.; et al. Efficient multi-party private set intersection against malicious adversaries. In Proceedings of the Proceedings of the 2019 ACM SIGSAC conference on cloud computing security workshop, 2019, pp. 93–104.
27. Pinkas, B.; Rosulek, M.; Trieu, N.; et al. PSIfrom PaXoS: Fast, malicious private set intersection. In Proceedings of the Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, 2020, pp. 739–767.
28. Orrù, M.; Orsini, E.; Scholl, P. Actively secure 1-out-of-n OT extension with application to private set intersection. In Proceedings of the Proceedings of Cryptographers' Track at the RSA Conference. Springer, 2017, pp. 381–396.
29. Rindal, P.; Schoppmann, P. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. IACR Cryptology ePrint Archive, 2021. https://eprint.iacr.org/2021/266.
30. Schoppmann, P.; Gascón, A.; Reichert, L.; et al. Distributed vector-OLE: Improved constructions and implementation. In Proceedings of the Proceedings of the 26th ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019, pp. 1055–1072.
31. Weng, C.; Yang, K.; Katz, J.; et al. Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits. Cryptology ePrint Archive, 2020. https://eprint.iacr.org/2020/925.
32. Egert, R.; Fischlin, M.; Gens, D.; Jacob, S.; Senker, M.; Tillmanns, J. Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters. *European Journal of Operational Research* **2015**, *139*, 371–389.
33. Ashok, V.; Mukkamala, R. A Scalable and Efficient Privacy Preserving Global Itemset Support Approximation Using Bloom Filters. In Proceedings of the IFIP Conference on Data and Applications Security and Privacy, 2014, pp. 382–389.
34. Debnath, S.; Dutta, R. Secure and Efficient Private Set Intersection Cardinality Using Bloom Filter. In Proceedings of the International Information Security Conference, 2015, pp. 209–226.
35. De Cristofaro, E.; Gasti, P.; Tsudik, G. Fast and Private Computation of Cardinality of Set Intersection and Union. In Proceedings of the CANS 2012. Springer, 2012, pp. 218–231.
36. Jarecki, S.; Liu, X. Fast Secure Computation of Set Intersection. In Proceedings of the SCN 2010. Springer, 2010, Vol. 6280, *Lecture Notes in Computer Science*, pp. 418–435.
37. Su, G.; Xu, M. A Survey on Secure Multi-party Computation Technology and Applications. *Information Communication Technologies and Policy* **2019**, pp. 19–22.

 doi:10.20944/preprints202405.0307.v1

38. Li, A. Research on Multi-party Statistical Computations Based on Functional Encryption. PhD thesis, Wuhan University of Technology, Wuhan, 2017.
39. Wang, H.; Dai, H.; Chen, S.; Chen, Z.; Chen, G. A Survey of Filter Data Structures. *Computer Science* **2024**, *51*, 35–40.
40. Yu, M.; Fabrikant, A.; Rexford, J. BUFFALO: Bloom filter forwarding architecture for large organizations. In Proceedings of the Proceedings of International Conference on Emerging Networking Experiments and Technologies, 2009, pp. 313–324.
41. Li, P.; Luo, B.; Zhu, W.; et al.. Cluster-based distributed dynamic cuckoo filter system for Redis. *International Journal of Parallel, Emergent and Distributed Systems* **2020**, *35*, 340–353.
42. Wang, F.; Chen, H.; Liao, L.; et al.. The power of better choice: Reducing relocations in cuckoo filter. In Proceedings of the Proceedings of International Conference on Distributed Computing Systems, 2019, pp. 358–367.
43. Gur, L.; Lis, D.; Dai, H.; et al.. Adaptive online cache capacity optimization via lightweight working set size estimation at scale. In Proceedings of the Proceedings of USENIX Annual Technical Conference, 2023, pp. 467–484.
44. Reviriego, P.; Martínez, J.; Larrabeiti, D.; et al.. Cuckoo Filters and Bloom Filters: Comparison and Application to Packet Classification. *IEEE Transactions on Network and Service Management* **2020**, *17*, 2690–2701.